

### REMARKS

Claims 10-27 are pending in the present application. Applicants have amended Claims 10, 12, 13, 15, 16, 19, 21, 22 and 24-26, and added Claim 27, herewith. Reconsideration of the claims is respectfully requested.

#### **I. 35 U.S.C. § 102, Anticipation**

The Examiner rejected Claims 10, 12, 13, 18, 19, 21, 22 and 26 under 35 U.S.C. § 102(e) as being anticipated by U.S. Pat. No. 5,848,423 to Ebrahim et al. This rejection is respectfully traversed.

With respect to Claim 10, such claim has been amended to include certain features from dependent Claim 12. Specifically, Claim 10 has been amended to emphasize that the claimed methodology is with respect to providing a security privilege for a method of a current thread. As amended, Claim 10 recites "searching the linked list for an entry having a stack frame pointer that matches the stack frame pointer of the method, wherein an entry of the linked list is a stack frame extension *comprising security privilege information for the method*". The cited reference does not teach any method for providing a security privilege for a method, and in particular does not teach any step of searching the linked list for an entry having a stack frame pointer that matches the stack frame pointer of the method, wherein an entry of the linked list is a stack frame extension *comprising security privilege information for the method*.

In rejecting Claim 12 (which included certain features now included in Claim 10), the Examiner cites Ebrahim figure 5/7 and Col. 9 Ln. 60 - 67 and Col. 10, Ln. 1-55 as teaching a stack frame extension comprising privilege information. Applicants show that there, Ebrahim states:

"Referring to FIGS. 5 and 7, each time the garbage collector begins a garbage collection root set traversal cycle, it first builds a list of root set object references using a root set locator procedure. For the purposes of this document, it is assumed that the root set object references in the CPU registers are easily located (280) using prior art techniques well known to those skilled in the art, and that this is done either before or after the search for object references in the program stack. A second assumption made here is that either all object references in the program stack are in method activation records as shown in FIG. 2, and/or that any

other object references in the program stack outside the activation records are located using prior art techniques well known to those skilled in the art, and therefore the task of finding any of those object references is not addressed in this document.

The root set locator procedure shown in FIG. 7 is repeated for each mutator task, each of which executes as a distinct thread. More particularly, each thread (mutator task) will have its own heap, stack, register values, and the heap for each thread is garbage collected independently of the others.

At step 280, the procedure locates all object references in the mutator task's registers, and adds them to the root set list.

The search for object references in the program stack begins at step 282 by locating the program stack for the mutator task, setting a pointer to the current frame in the program stack, which is the last frame on that program stack. If there are no activation records on the program stack, the procedure exits.

Otherwise, at step 286 the aforementioned hash function is applied the invocation address in the activation record of the stack frame currently being inspected by the root set locator procedure to generate a pointer to a hash table entry (286). Then the hash table entries starting at the address generated at step 286 are searched until a hash table entry matching the invocation address in the activation record is found (288). For the large majority of activation records in the stack, the first hash table entry checked at step 288 will match the activation record's invocation address.

Next, the information in the located hash table entry is used to determine the number of object references (NRefs), if any, in the activation record being processed, and the location of the first object reference offset in the offset table (290). If the hash table entry indicates that the activation record contains no object references, the processing of the activation record is complete and the locator procedure continues by looking for an activation record in the next earlier stack frame in the stack (284), which is located using the previous stack frame pointer in the stack frame last processed by the procedure.

If the hash table entry indicates that the activation record contains at least one object reference, the NRef object reference offsets stored in the offset table 210 for the activation record are used to locate the NRef object references in the activation record 230 being processed by the root set locator procedure. The located NRef object references are added by the root set locator procedure to the root set list (292). Then the root set locator procedure resumes processing at step 284, where it looks for the next earlier activation record in the stack. If the last activation record processed was the first one in the program stack, the procedure exits. Otherwise processing the next earlier activation record continues at step 286."

As can be seen, this passage describes a technique for obtaining object references maintained in activation records, in order to build a list of root set object references for use by a garbage collector routine. This passage has nothing to do with providing any type of privilege or security privilege for a method of a current thread, as claimed. Thus, it is shown that the cited reference does not anticipate amended Claim 10.

Applicants traverse the rejection of Claims 12 and 13 for similar reasons to those given above regarding Claim 10.

With respect to Claim 18, the cited reference does not teach all claimed elements of the recited stack frame extensions. For a prior art reference to anticipate in terms of 35 U.S.C. 102, every element of the claimed invention must be identically shown in a single reference. *In re Bond*, 910 F.2d 831, 15 USPQ2d 1566 (Fed. Cir. 1990). Applicants urge that the cited reference does not teach a stack frame extension comprising *a data field for privilege data for the method*. In rejecting Claim 18, the Examiner cites Ebrahim figure 5/7 and Col. 9 Ln. 60 – 67 and Col. 10, Ln. 1-55 as teaching this claimed feature.

Applicants show that there, Ebrahim states:

“Otherwise, at step 286 the aforementioned hash function is applied the invocation address in the activation record of the stack frame currently being inspected by the root set locator procedure to generate a pointer to a hash table entry (286). **Then the hash table entries starting at the address generated at step 286 are searched until a hash table entry matching the invocation address in the activation record is found (288).** For the large majority of activation records in the stack, the first hash table entry checked at step 288 will match the activation record's invocation address.

Next, the information in the located hash table entry is used to **determine the number of object references (NRefs), if any, in the activation record being processed, and the location of the first object reference offset in the offset table (290).** If the hash table entry indicates that the activation record contains no object references, the processing of the activation record is complete and the locator procedure continues by looking for an activation record in the next earlier stack frame in the stack (284), which is located using the previous stack frame pointer in the stack frame last processed by the procedure.” (emphasis added by Applicants)

As can be seen, two things are determined from a matching hash table entry – (1) the number of object references in the activation record, and (2) the location of the first

object reference offset in the offset table. Neither of these items can reasonably be construed to teach a stack frame extension comprising *a data field for privilege data for the method*. These two items are also shown by Ebrahim FIG. 5, elements Nrefs and Ptr. As described at Ebrahim Col. 9, lines 26-35, the data stored in the hash table includes the invocation address (IA) and the number of object references (Nrefs) in the activation record. If the number of object references in the activation table is not zero, a pointer (Ptr) is included to point to an entry in an object reference offset table. An object reference is defined by Ebrahim at Col. 2, lines 5-14 to be data structure that includes a pointer to an object/data structure. Thus, the number of object references as taught by Ebrahim is just that, a number. This number provides no type of privilege data/information for a method, as claimed. Similarly, the Ptr pointer in this hash table provides no type of privilege data/information for a method, but rather is a pointer or address to an object reference offset table. These object reference offsets are used by a root set locator procedure to add each of the object references in the activation record into a root set list for garbage collection (Col. 5, lines 53-55), and these offsets/pointers/addresses have nothing to do with privileges for a method, as claimed.

In summary, the Ebrahim hash table which is searched for a matching invocation address includes three types of items - (1) invocation addresses, (2) raw numbers indicated the number of object references for a given activation record, and (3) addresses/pointers to another table. None of these items teach a stack frame extension comprising *a data field for privilege data for the method*, as claimed. Thus, it is shown that Claim 18 is not anticipated by the cited reference.

Applicants traverse the rejection of Claims 19, 21, 22 and 26 for similar reasons to those given above regarding Claim 10.

Therefore, the rejection of Claims 10, 12, 13, 18, 19, 21, 22 and 26 under 35 U.S.C. § 102 has been overcome.

## II. 35 U.S.C. § 103, Obviousness

The Examiner rejected Claims 11, 14, 20 and 23 under 35 U.S.C. § 103(a) as being unpatentable over U.S. Pat. No. 5,848,423 to Ebrahim et al. in view of U.S. Pat. No. 6,493,816 B1 to Munroe et al. This rejection is respectfully traversed.

Applicants initially traverse this rejection for similar reasons to those given above regarding Claims 10 and 19, of which these claims depend upon.

Further with respect to Claim 14 (and similarly for Claim 23), Applicants show that none of the cited references teach or suggest the claimed feature of "adding an entry to the linked list if no matching entries are found in response to a request to enable a privilege for the method". In rejecting Claim 14, the Examiner states that "Munroe teaches the process of Claim 10 further comprising: adding an entry to the linked list if no matching entries are found in response to a request to enable a privilege for the method (Col. 13 Ln. 1-10)". Applicants show that there, Munroe states:

"In particular, the IRO Manager cache is preferably implemented as a hashtable-stack hybrid, with IROs indexed in the IRO Manager cache by frame. **When a new IRO is created, it is placed in the IRO Manager cache under the current frame with other IROs that have been created under that frame.** These IROs remain in the IRO Manager cache until that frame is exited. When the JVM exits its frame all the cached IROs created within that frame are removed from the cache, but cached IROs created within previous frames remain in the IRO Manager cache." (emphasis added by Applicants)

As can be seen, this passage teaches placing the IRO Manager cache under the current frame *when a new IRO is created*. Thus, this step is responsive to creation of a new IRO. In contrast, Claim 14 is directed to an entry adding step *that is responsive to a request to enable a privilege for a method*. As described by Munroe at Col. 12, lines 25-32, these IROs are created for each java object that is accessed by a Java virtual machine (JVM). IROs for transient objects are created whenever the virtual machine informs the IRO manager that it has created an object (Col. 14, lines 21-23) and IROs for persistent objects are created whenever address translations from shared address space to native addresses is requested (Col. 14, lines 39-41), and thus such IRO creation and associated cache entry addition to a linked list has nothing to do with, and in particular is not responsive to, requests to enable a privilege for a method, as claimed. Thus, Claim 14 (and similarly for Claim 23) is further shown to not be obvious in view of the cited references.

Therefore, the rejection of Claims 11, 14, 20 and 23 under 35 U.S.C. § 103 has been overcome.

**III. Allowable Subject Matter**

Applicants graciously acknowledge the allowance of Claim 17.

In addition, the Examiner stated that Claims 15, 16, 24 and 25 were objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. In response, the claims have been rewritten accordingly to overcome this objection.

**IV. Newly Added Claim**

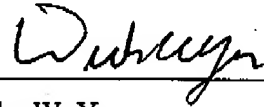
Claim 27 has been added herewith. Examination of such claim is respectfully requested.

**V. Conclusion**

It is respectfully urged that the subject application is patentable over the cited references and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: 7/7/04

Respectfully submitted,



Duke W. Yee  
Reg. No. 34,285  
Wayne P. Bailey  
Reg. No. 34,289  
Yee & Associates, P.C.  
P.O. Box 802333  
Dallas, TX 75380  
(972) 367-2001  
Attorneys for Applicants